

National University

of ScienceS & Technology

CS381-Network Security

Assignment 1

Submitted To: Prof. Sana Qadir

Submitted By:

|  |  |  |
| --- | --- | --- |
| No | Name | CMS |
| 1 | Asad Ur Rehman | 371077 |
| 2 | Zohair Shakeel | 365721 |
| 3 | Muhammad Zubair Khalid | 365738 |
| 4 | Rana Mahad Ahmer | 367679 |
| 5 | Misbah Juwayriyyah | 382416 |

Submission Date: 9-10-2024

# **Assignment:**

## **Task:**

Create software that can encrypt and decrypt using S-AES.

Test data: A binary plaintext of 0110 1111 0110 1011 encrypted with a binary key of 1010 0111 0011 1011 should give a binary ciphertext of 0000 0111 0011 1000. Decryption should work correspondingly.

## **Solution:**

The Simplified AES uses two rounds of common AES steps. It encrypts and decrypts 16-bit binary data using a 16-bit key. Key features include:

1. **Substitution (S-Box)**: Non-linear substitution of bits.
2. **Shift Rows**: Shifting positions of bits in the state.
3. **Mix Columns**: Mixing bits from different columns.
4. **Add Round Key**: Combining the state with the key using XOR.

### **Note:**

The code was written in Kotlin, Kotlin project is attached in Code.rar zip file, GitHub link is also provided at the end of this file.

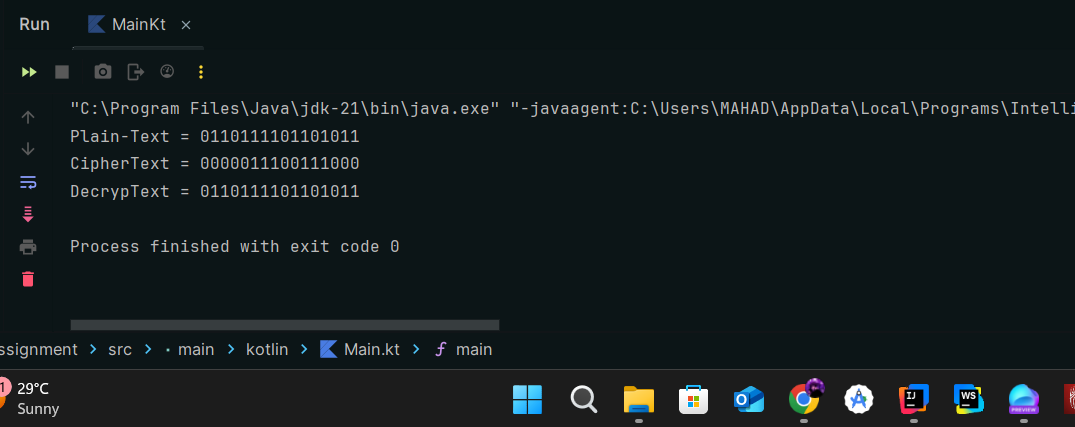
### **Code:**

|  |
| --- |
| val *sBox* = *arrayOf*(  0b1001, 0b0100, 0b1010, 0b1011,  0b1101, 0b0001, 0b1000, 0b0101,  0b0110, 0b0010, 0b0000, 0b0011,  0b1100, 0b1110, 0b1111, 0b0111 ) val *sBoxI* = *arrayOf*(  0b1010, 0b0101, 0b1001, 0b1011,  0b0001, 0b0111, 0b1000, 0b1111,  0b0110, 0b0000, 0b0010, 0b0011,  0b1100, 0b0100, 0b1101, 0b1110 )  fun subWord(word: Int): Int {  return (*sBox*[word shr 4] shl 4) + *sBox*[word and 0x0F] }  fun rotWord(word: Int): Int {  return ((word and 0x0F) shl 4) + ((word and 0xF0) ushr 4) }  fun keyExpansion(key: Int): Triple<List<Int>, List<Int>, List<Int>> {  val cCon1 = 0x80  val rCon2 = 0x30  val w = *arrayOfNulls*<Int>(6)  w[0] = (key and 0xFF00) ushr 8  w[1] = key and 0x00FF  w[2] = w[0]!! xor (*subWord*(*rotWord*(w[1]!!)) xor cCon1)  w[3] = w[2]!! xor w[1]!!  w[4] = w[2]!! xor (*subWord*(*rotWord*(w[3]!!)) xor rCon2)  w[5] = w[4]!! xor w[3]!!  return Triple(  *intToState*((w[0]!! shl 8) + w[1]!!),  *intToState*((w[2]!! shl 8) + w[3]!!),  *intToState*((w[4]!! shl 8) + w[5]!!)  ) }  fun gfMult(a: Int, b: Int): Int {  var product = 0  var aa = a and 0x0F  var bb = b and 0x0F  while (aa != 0 && bb != 0) {  if (bb and 1 != 0) {  product = product xor aa  }  aa = aa shl 1  if (aa and (1 shl 4) != 0) {  aa = aa xor 0b10011  }  bb = bb ushr 1  }  return product }  fun intToState(n: Int): List<Int> {  return *listOf*(n shr 12 and 0xF, (n shr 4) and 0xF, (n shr 8) and 0xF, n and 0xF) }  fun stateToInt(m: List<Int>): Int {  return (m[0] shl 12) + (m[2] shl 8) + (m[1] shl 4) + m[3] }  fun addRoundKey(s1: List<Int>, s2: List<Int>): List<Int> {  return s1.*zip*(s2).*map* **{** (i, j) **->** i xor j **}** }  fun subNibbles(sbox: Array<Int>, state: List<Int>): List<Int> {  return state.*map* **{** sbox[**it**] **}** }  fun shiftRows(state: List<Int>): List<Int> {  return *listOf*(state[0], state[1], state[3], state[2]) }  fun mixColumns(state: List<Int>): List<Int> {  return *listOf*(  state[0] xor *gfMult*(4, state[2]),  state[1] xor *gfMult*(4, state[3]),  state[2] xor *gfMult*(4, state[0]),  state[3] xor *gfMult*(4, state[1])  ) }  fun inverseMixColumns(state: List<Int>): List<Int> {  return *listOf*(  *gfMult*(9, state[0]) xor *gfMult*(2, state[2]),  *gfMult*(9, state[1]) xor *gfMult*(2, state[3]),  *gfMult*(9, state[2]) xor *gfMult*(2, state[0]),  *gfMult*(9, state[3]) xor *gfMult*(2, state[1])  ) }  fun encrypt(plaintext: Int, key: Int): Int {  val (preRoundKey, round1Key, round2Key) = *keyExpansion*(key)  var state = *addRoundKey*(preRoundKey, *intToState*(plaintext))  state = *mixColumns*(*shiftRows*(*subNibbles*(*sBox*, state)))  state = *addRoundKey*(round1Key, state)  state = *shiftRows*(*subNibbles*(*sBox*, state))  state = *addRoundKey*(round2Key, state)  return *stateToInt*(state) }  fun decrypt(ciphertext: Int, key: Int): Int {  val (preRoundKey, round1Key, round2Key) = *keyExpansion*(key)  var state = *addRoundKey*(round2Key, *intToState*(ciphertext))  state = *subNibbles*(*sBoxI*, *shiftRows*(state))  state = *inverseMixColumns*(*addRoundKey*(round1Key, state))  state = *subNibbles*(*sBoxI*, *shiftRows*(state))  state = *addRoundKey*(preRoundKey, state)  return *stateToInt*(state) }  fun main() {  val plaintext = 0b0110111101101011  val key = 0b1010011100111011  val ciphertext = *encrypt*(plaintext, key)   *println*("Plain-Text = ${plaintext.*toString*(2).*padStart*(16, '0')}")  *println*("CipherText = ${ciphertext.*toString*(2).*padStart*(16, '0')}")  val decrypted = *decrypt*(ciphertext, key)  *println*("DecrypText = ${decrypted.*toString*(2).*padStart*(16, '0')}") } |

### **Screenshots:**

|  |
| --- |
|  |

### **Output:**



#### **TEXT:**

**Plain-Text = 0110111101101011**

**CipherText = 0000011100111000**

**DecrypText = 0110111101101011**

### **GitHub Link:**

<https://github.com/RanaMahadAhmer/NS_Assignment_1>